

**Содержание**

[Реферат 5](#_Toc185778482)

[Введение 6](#_Toc185778483)

[1 Постановка задачи 7](#_Toc185778484)

[2 Теоретическая часть задания 8](#_Toc185778485)

[3 Описание алгоритма программы 9](#_Toc185778486)

[4 Описание программы 13](#_Toc185778487)

[5 Тестирование 15](#_Toc185778488)

[Заключение 21](#_Toc185778489)

[Список используемой литературы 22](#_Toc185778490)

[Приложение А. Листинг программы 23](#_Toc185778491)

Реферат

Отчёт 44 стр, 11 рисунков

ГРАФ, ОБХОД В ШИРИНУ, АЛГОРИТМ ПОИСКА КРАТЧАЙШЕГО РАССТОЯНИЯ

Цель исследования – разработка алгоритма нахождения кратчайшего пути от одного объекта до другого для игры Лабиринт.

В работе рассмотрен обход графа в ширину. Благодаря данному алгоритму ищется кратчайшее расстояние от одной вершины графа до другой и сразу записывается путь, состоящий из набора связанных между собой вершин.

Также программа включает себя алгоритм создания графа на основе матрицы лабиринта, состоящей из разных объектов.

Введение

Алгоритм обхода в ширину — это один из основных методов поиска в графах. Он используется для обхода всех вершин графа, начиная с заданной. Алгоритм работает по принципу уровней: он сначала исследует все вершины, которые находятся на расстоянии одного ребра от начальной, затем — все вершины, находящиеся на расстоянии двух ребер, и так далее. Это позволяет находить кратчайшие пути в невзвешенных графах.

Алгоритм реализуется с помощью очереди. Сначала в очередь помещается стартовая вершина. Затем, пока очередь не опустеет, извлекается вершина, и в неё добавляются все её смежные, ещё не посещенные вершины. Каждая посещенная вершина помечается, что предотвращает повторные посещения. Это делает обход эффективным для поиска в графах и позволяет убедиться, что каждая вершина будет изучена ровно один раз.

Алгоритм обхода в ширину находит применение в различных областях, таких как поиск в интернете, планирование маршрутов и игры. В теории графов он используется для нахождения компонент связности и проверки двудольности графа. Также поиск в ширину применяется в задачах, требующих нахождения минимального количества шагов, необходимых для достижения определенной вершины, такие как игры и головоломки.

Для реализации была выбрана среда разработки IntelliJ IDEA, язык программирования – Java.

Целью данной курсовой работы является разработка программы на Java, который является высокоуровневым и широко используемым языком. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в ширину, осуществляющий поиск кратчайшего пути в лабиринте.

1 Постановка задачи

Создать программу, в которой будет реализован игровой агент, ищущий кратчайший путь до игрока.

Реализовать несколько сущностей с методами движения для игры лабиринт.

Разработать функцию обрабатывающую файл с массивом лабиринта для его построения и вывода на экран.

Реализовать графический интерфейс для более удобного отображения игры.

Создать метод поиска кратчайшего пути и выбор следующей клетки, на которую должен пойти игровой агент.

Установить условия завершения игры для победы и поражения.

Предусмотреть возможность сохранения результата игры.

Разработать контроллер мыши и клавиатуры для управления движением игрока Задания выполняются в соответствии с вариантом №31.

2 Теоретическая часть задания

Граф G представляет собой набор вершин X1, X2, ..., Xn и множество рёбер, соединяющих эти вершины.

Список смежности — это один из способов представления графа, который позволяет эффективно хранить информацию о его вершинах и рёбрах. В этом представлении каждая вершина графа ассоциируется со списком, содержащим все её смежные вершины. Это позволяет экономить память, особенно в разреженных графах, где количество рёбер значительно меньше, чем квадрат числа вершин.

Игровой агент в игре лабиринт — это объект или программа, задача которой заключается в нахождении оптимального пути через лабиринт. Обычно агент может быть представлен как человекоподобный персонаж или алгоритм, способный взаимодействовать с игровым миром.

3 Описание алгоритма программы

Алгоритм для реализации игрового агента в лабиринте включает несколько ключевых этапов, которые помогут агенту находить путь до цели.

На основе построенного лабиринта создаются списки смежности для графа. Каждая вершина включает в себя необходимые поля: **int x** и **int y** – для определения какой клетке принадлежит вершина, **Vertex next** – для обхода каждой вершины, даже изолированных, **Vertex[] adjVerts** – массив смежных вершин, **int distId** – для записи расстояния до вершины и **Vertex[] shortPath** **–** для записи пути к данной вершине.

После идет выполнение функции поиска кратчайшего пути на основе обхода в ширину для определения следующего хода для каждого игрового агента.

При этом игровые агенты не следуют друг за другом, а идут в обход (также по кратчайшему пути), чтобы поймать игрока.

Ниже представлен псевдокод функций «createGraph», «findNextWay»

**createGraph(Map map)**

1. Функция createGraph(Карта map) возвратит Vertex

2. массив array = map.getMap()

3. целое k = 0

4. Для целого i от 1 до длины array (не включая) делай

5. Для целого j от 1 до длины array[i] (не включая) делай

6. Если array[i][j] не равно WALL\_CHARACTER тогда

7. k = k + 1

8. массив вертексов verts = новый Vertex[k]

9. k = 0

10. Для целого i от 1 до длины array (не включая) делай

11. Для целого j от 1 до длины array[i] (не включая) делай

12. Если array[i][j] не равно WALL\_CHARACTER тогда

13. verts[k] = новый Vertex(j, i, null, k)

14. k = k + 1

15. Для целого i от 0 до длины verts (не включая) делай

16. Если i не равно длине verts - 1 тогда

17. verts[i].next = verts[i+1]

18. Для целого i от 0 до длины verts (не включая) делай

19. Vertex vert = verts[i]

20. массив adj = новый Vertex[4]

21. k = 0

22. целое x = vert.x

23. целое y = vert.y

24. Если array[y][x-1] не равно WALL\_CHARACTER тогда

25. Vertex temp = найти Vertex из verts, где n.y == y и n.x == x-1

26. adj[k] = temp

27. k = k + 1

28. Если array[y-1][x] не равно WALL\_CHARACTER тогда

29. Vertex temp = найти Vertex из verts, где n.y == y-1 и n.x == x

30. adj[k] = temp

31. k = k + 1

32. Если array[y][x+1] не равно WALL\_CHARACTER тогда

33. Vertex temp = найти Vertex из verts, где n.y == y и n.x == x+1

34. adj[k] = temp

35. k = k + 1

36. Если array[y+1][x] не равно WALL\_CHARACTER тогда

37. Vertex temp = найти Vertex из verts, где n.y == y+1 и n.x == x

38. adj[k] = temp

39. k = k + 1

40. vert.adjVerts = скопировать массив adj до k

41. Вернуть verts[0]

**findNextWay(Player player, Enemy enemy, List<Enemy> otherEnemies)**

1. Функция найтиСледующийПуть (Игрок игрок, Враг враг, Список<Враг> другиеВраги)

2. Вершина временная = текущая\_вершина

3. Целое k = 0

4. Пока временная не равно null

5. временная.кратчайшийПуть = null

6. k = k + 1

7. временная = временная.следующая

8. временная = текущая\_вершина

9. Целый\_массив расстояния = новый Целый\_массив[k]

10. Заполнить\_массив(расстояния, -1)

11. временная = текущая\_вершина

12. Пока временная не равно null

13. Если временная.x равно враг.getXPosition() И временная.y равно враг.getYPosition()

14. прервать\_цикл

15. временная = временная.следующая

16. Если игрок.мертв

17. вернуть временная

18. Массив<Вершина> посещенные\_вершины = null

19. Очередь<Вершина> очередь = новая Очередь<Вершина>()

20. Добавить\_в\_очередь(очередь, временная)

21. посещенные\_вершины = новый Массив<Вершина>[1]

22. посещенные\_вершины[0] = временная

23. Пока очередь не пуста

24. Вершина новая\_вершина = извлечь\_из\_очереди(очередь)

25. Если новая\_вершина.смежные\_вершины не равно null

26. Для каждого i от 0 до новая\_вершина.смежные\_вершины.длина - 1

27. Вершина вершина = новая\_вершина.смежные\_вершины[i]

28. Если ни\_одна\_из(посещенные\_вершины, n -> n == вершина) И ни\_один\_из(другиеВраги, n -> n.столкновение(вершина.x, вершина.y))

29. Добавить\_в\_очередь(очередь, вершина)

30. Массив<Вершина> временные\_вершины = новый Массив<Вершина>[посещенные\_вершины.длина + 1]

31. временные\_вершины = скопировать\_массив(посещенные\_вершины, посещенные\_вершины.длина + 1)

32. временные\_вершины[посещенные\_вершины.длина] = вершина

33. посещенные\_вершины = временные\_вершины

34. расстояния [вершина.id\_расстояния] = расстояния [новая\_вершина.id\_расстояния] + 1

35. Массив<Вершина> кратчайший\_путь

36. Если новая\_вершина.кратчайшийПуть не равно null

37. кратчайший\_путь=скопировать\_массив (новая\_вершина.кратчайшийПуть , новая\_вершина.кратчайшийПуть.длина + 1)

38. Иначе

39. кратчайший\_путь = новый Массив<Вершина>[1]

40. кратчайший\_путь[кратчайший\_путь.длина - 1] = вершина

41. вершина.кратчайшийПуть = кратчайший\_путь

42. временная = текущая\_вершина

43. Пока временная не равно null

44. Если временная.x равно игрок.getXPosition() И временная.y равно игрок.getYPosition()

45. прервать\_цикл

46. временная = временная.следующая

47. Если временная.кратчайшийПуть равно null

48. вернуть null

49. вернуть временная.кратчайшийПуть[0]

50. Конец\_функции

4 Описание программы

Для реализации данной программы был выбран язык программирования Java. Это высокоуровневый и универсальный язык программирования. Программа представляет собой оконное приложение, разработанное в среде разработки IntelliJ IDEA.

Программа включает в себя несколько классов, которые четко структурированы в различных папках проекта. Папка config включает в себя файл Configuration, отвечающий за все константы проекта. Папка controller хранит в себе классы MouseController и MovementController, которые необходимы для управления интерфейсом пользователем. Также есть папка сущностей (domen). В ней находятся 2 абстрактных класса (DynamicObject и GameObject), от которых наследуются другие сущности (Enemy, Exit, Map, Player, Entity, Coin). Для реализации игрового процесса был добавлен класс GameMaster, находящийся в папке service. Кроме того, util содержит вспомогательные классы(Vertex и FileUtils) для работы с графом и файлами. Существует также класс Main, который создает класс GameFrame, где постоянно вызывается метод paint(Graphics graphics), в свою очередь вызывающий метод renderFrame(Graphics graphics) из класса GameMaster для отрисовки самой картинки. Также есть папка resources, в которой хранятся все необходимые изображения, результаты игры и тп.

Работа программы начинается с того, что создается окно, в котором обрисовывается меню игры.

Затем пользователю необходимо выбрать: начать играть или выйти из игры. После выбора «начать играть» начинается создание лабиринта из уже созданного файла map.ber. Программа проверяет обязательное наличие стен по краям лабиринта, наличие входа, врагов и самого игрока. Если что-то отсутствует, то программа завершает свою работу.

После успешной загрузки карты на основе неё заполняется массив объектов, то есть создаются объекты классов Coin, Player, Exit, Enemy. Далее формируются списки смежности.

Когда вся подготовка закончена, начинается отрисовка игры. Места, где в файле map.ber помечались как значение константы из файла WALL\_CHARACTER, обрисовывается как картинка стены из папки resources, значение константы COIN\_CHARACTER – как картинка монеты из той же папки, и так ко всем символам.

Сразу как обрисовывается игра игрок может начинать двигаться, нажимая на стрелки на клавиатуре (UP, DOWN, LEFT, RIGHT). Как только игрок нажимает на одну из клавиш – происходит вызов метода класса move(Direction direction), в котором проверяется наличие доступного для прохода места. Если доступно, то позиция игрока меняется.

Враги тоже начинают двигаться, но только совершают свой ход через каждые 30 кадров. Сначала вызывается метод findNextWay(Player player, Enemy enemy, List<Enemy> otherEnemies), в котором ищется кратчайший путь до игрока, исключая следование друг за другом игровых агентов, и возвращается координаты позиции на следующий ход. Затем совершается тот же метод move(Direction direction), с той же проверкой на свободное место.

Задача игрока – собрать все монеты в лабиринте и выйти из него через выход. Как только игрок двигается и вызывается метод move(Direction direction) происходит проверка на коллизию с монетами, врагами и выходом (метод collision(int x, int y)). При коллизии с монетой – она удаляется и на счёт игрока добавляется одна монета. В то же время при коллизии с выходом – игра завершается, но если игрок не собрал все монеты, то выход не откроется и на его клетку нельзя будет зайти. Кроме того, если игрок столкнется с игровым агентом, то он проиграет и будет выведено соответствующее сообщение. Время игры также рассчитывается и записывается в файл result.txt в папке resourses.

После любого исхода: победы или поражения – появляется соответствующее сообщение и через определенное время возвращается то же начальное меню, где можно начать игру заново, либо выйти из игры.

5 Тестирование

Среда разработки IntelliJ IDEA предоставляет все средства необходимые для тестирования и отладки программы. Во время тестирования можно не только проходить через каждую строку программы, но и менять значения полей.

Тестирование проводилось в процессе разработки и после завершения работы над программой. Благодаря этому были найдены ошибки в коде, недочеты, а так же условия, которые необходимо было учесть во время игрового процесса.

Ниже продемонстрирована работа программы. На рисунках 2-4 представлен игровой процесс. Рисунок 5 показывает обход с другой стороны другим игровым агентом для более успешной охоты на игрока. Рисунок 6 говорит о том, что игрок был пойман одним из игровых агентов. На рисунке 7 показан возврат в меню после поражения, а рисунок 8 иллюстрирует запись результата в файл. На рисунке 9 можно заметить, что все монеты собраны и теперь можно выйти через определенную клетку и выиграть, что и показано на рисунке 10. Рисунок 11 говорит нам о том, что результат победы тоже записывается.

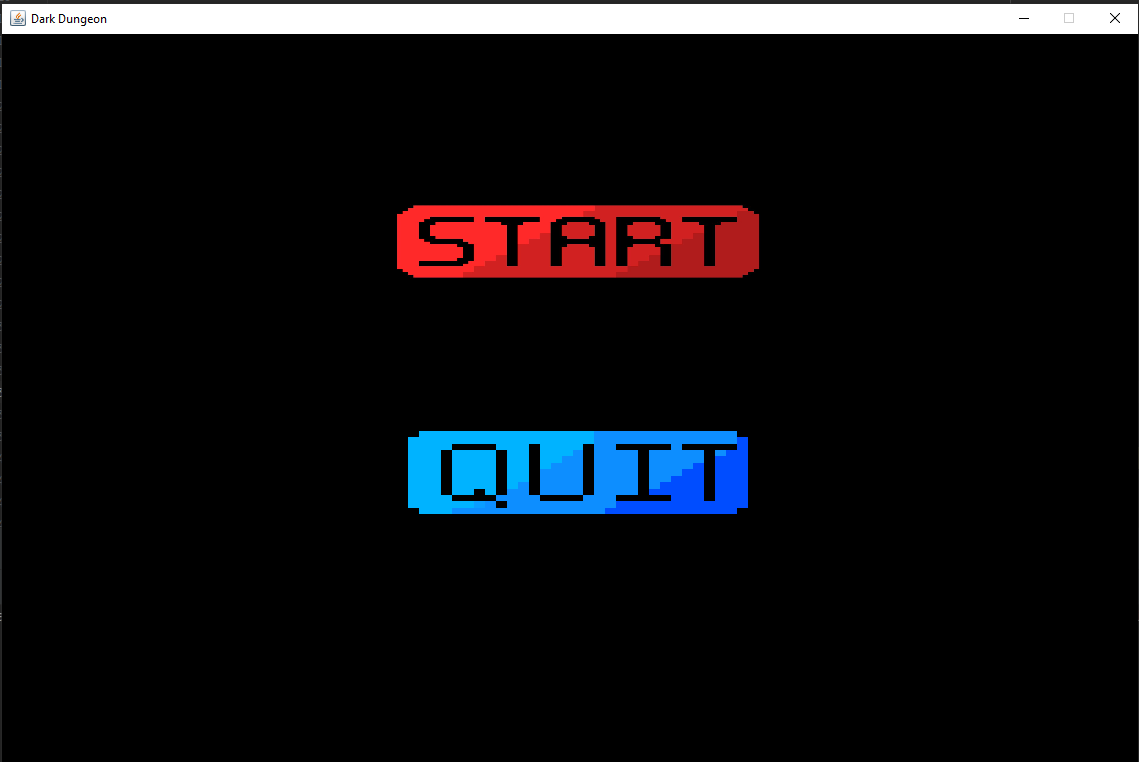


Рисунок 1 - Меню программы



Рисунок 2 - Старт игры



Рисунок 3 - Собраны 4 монеты



Рисунок 4 - Собраны ещё несколько монет

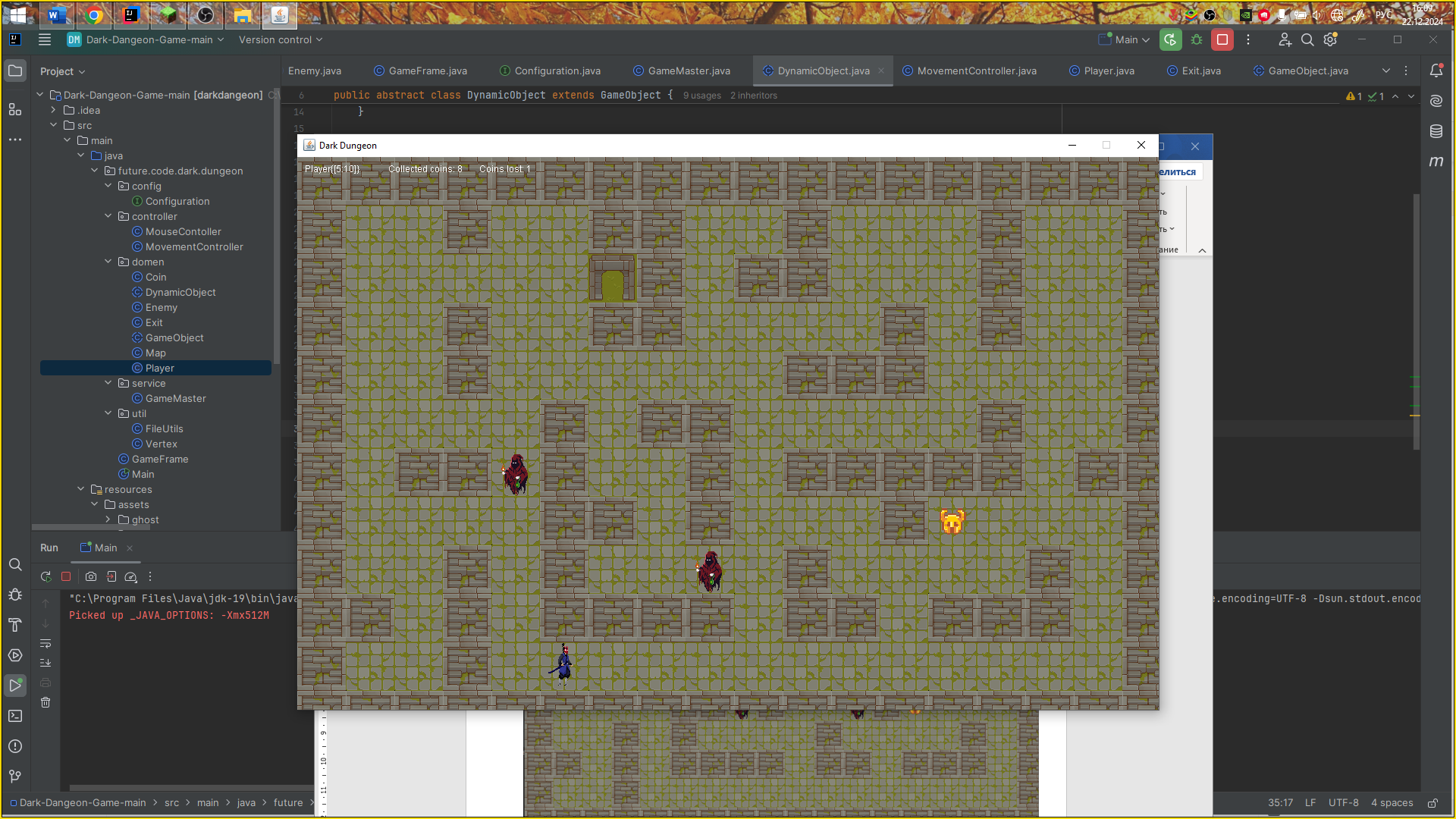


Рисунок 5 - Пример того, что игровые агенты идут в окружение



Рисунок 6 - Игровой агент поймал игрока

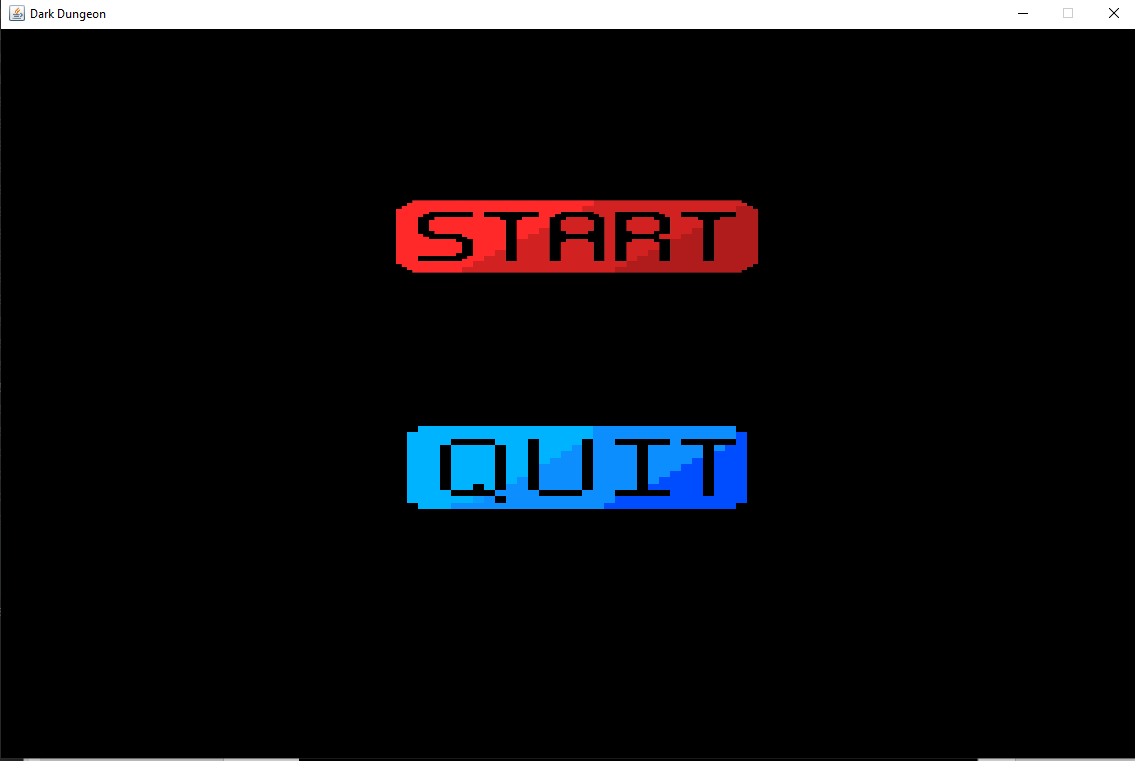


Рисунок 7 - Возврат в меню



Рисунок 8 - Запись результата в файл

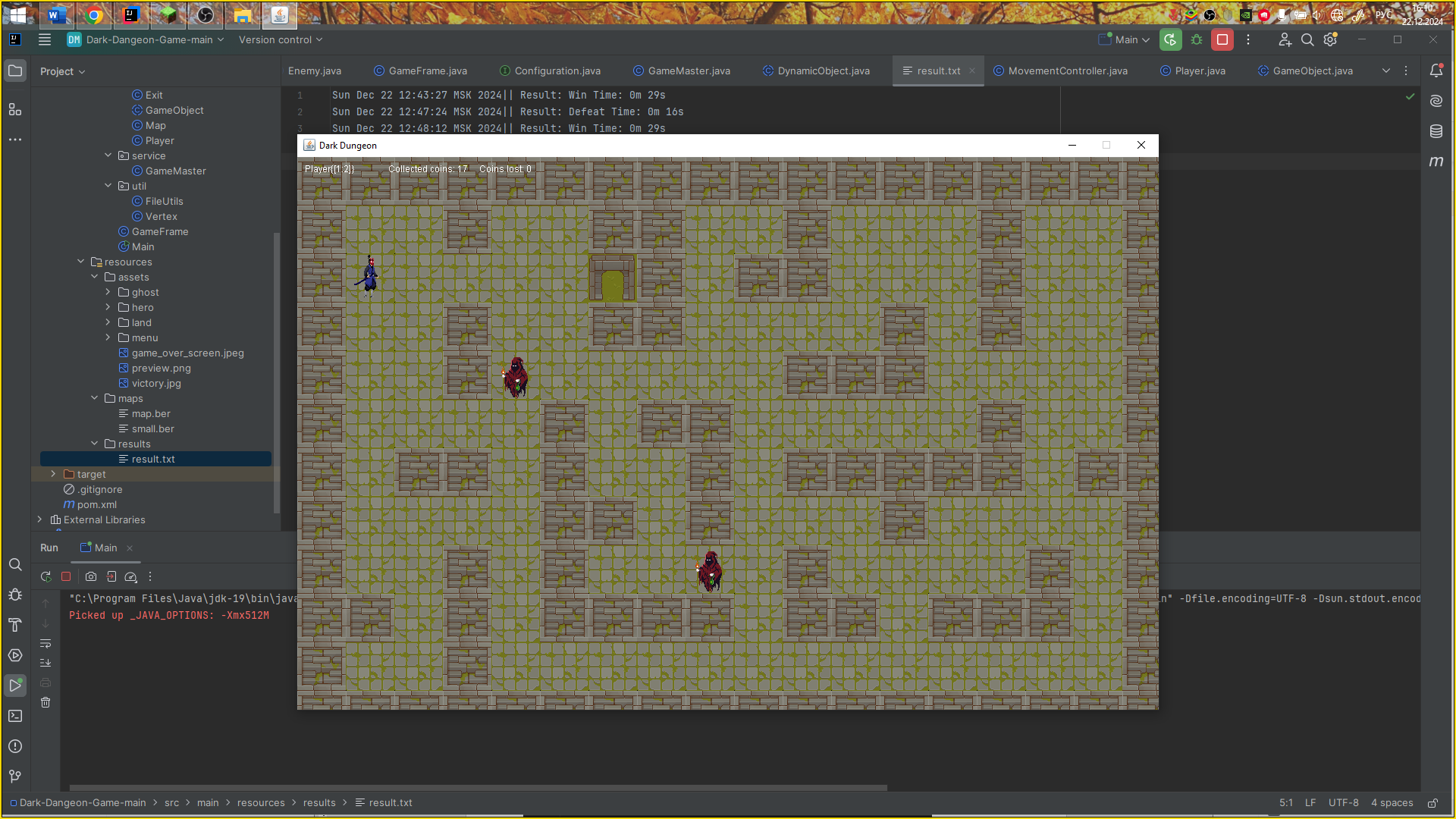


Рисунок 9 - Все монеты собраны

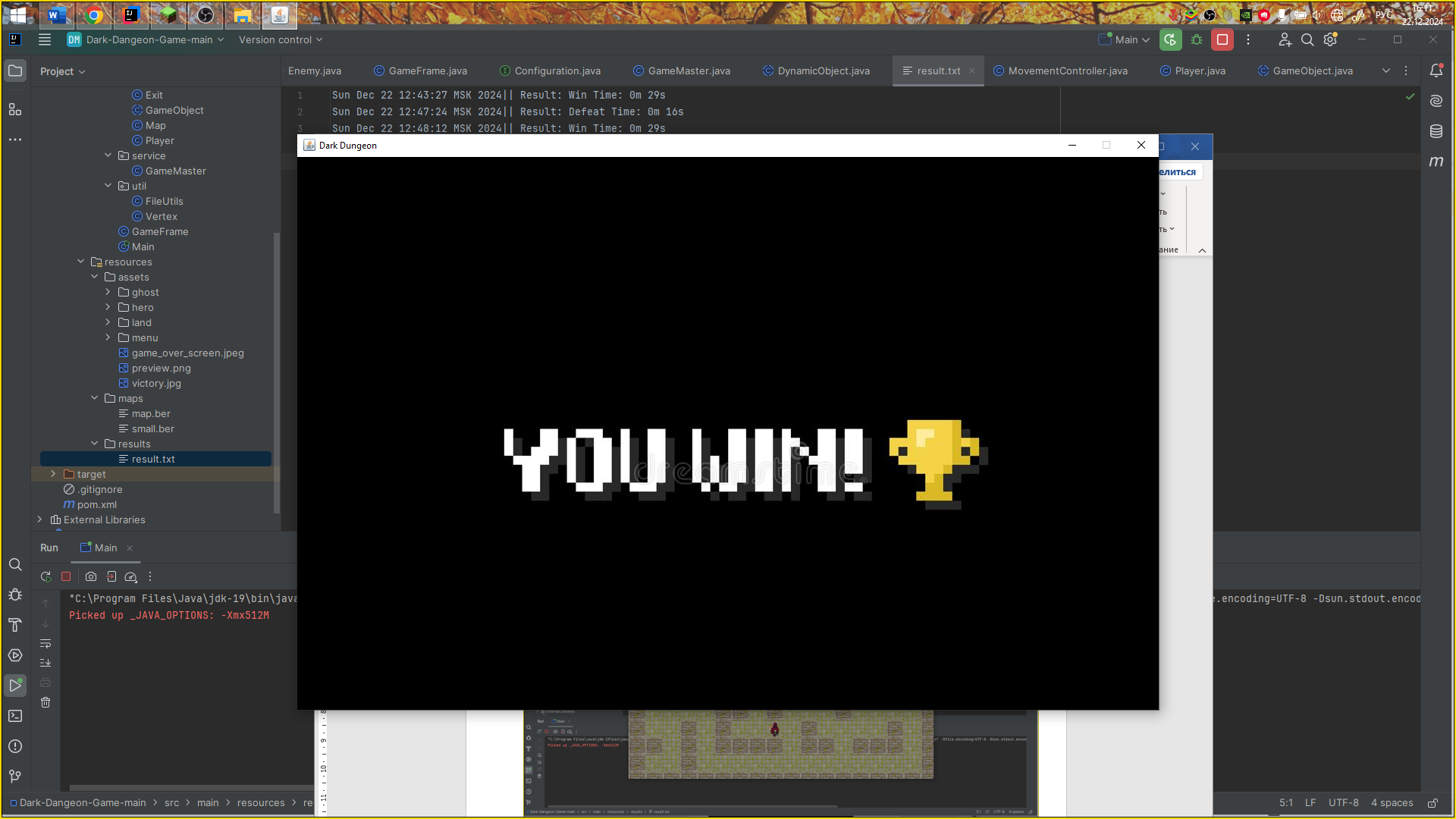


Рисунок 10 - Игрок достиг выхода и выиграл



Рисунок 11 - Запись результата в файл

Заключение

В ходе выполнения курсового проекта "Разработка игрового агента для игры лабиринт" было достигнуто несколько ключевых целей, которые способствовали пониманию алгоритмов поиска и реализации искусственного интеллекта в игровом пространстве.

Созданный агент продемонстрировал способности к эффективной навигации по лабиринту, используя методы поиска пути. Это не только подтвердило теоретические знания, но и позволило на практике овладеть принципами оптимизации поиска.

Кроме того, проект способствовал развитию навыков программирования, работы с графическими интерфейсами и тестирования ПО. Внедрение дополнительных функций, таких как визуализация пути и возможность выбора различных стратегий поиска, увеличило интерактивность и интерес к проекту.

Таким образом, результаты работы по разработке игрового агента подтверждают актуальность изучения искусственного интеллекта в игровой индустрии и открывают новые горизонты для дальнейших исследований и разработок. Будущие работы могут сосредоточиться на улучшении алгоритмов, включая машинное обучение для более адаптивного поведения игрового агента.

Список используемой литературы

1. Ричард С. Секунд. Искусственный интеллект: современные подходы. – М.: Вильямс, 2019.

2. Хорн, М. Алгоритмы и структуры данных для игр. – СПб: БХВ-Петербург, 2018.

3. Сэм Коттлер. Разработка игр: от идеи до реализации. – М.: Питер, 2020.

4. Л. Д. Зенкевич. Методы поиска в графах и их применение. – М.: Научный мир, 2021.

5. Д. Клас. Глубокое обучение для игр и разработка искусственного интеллекта. – М.: Дорлинг Киндерсли, 2022.

6. Ли, Э. Искусственный интеллект для игр: практическое руководство. – М.: БХВ-Петербург, 2017.

7. Бельтюков, А. Основы разработки игр: теория и практика. – СПб: Питер, 2023.

8. К. С. К. Л. Алгоритм A: Быстрый и эффективный способ поиска пути. – Научно-технический журнал, 2020.

9. П. Паттерсон. Поиск пути и алгоритмы: примеры и решения. – М.: Аспект Пресс, 2019.

10. Хеллер, А. Игровые агенты: основанные на правилах и адаптивные системы. – М.: Издательство МГУ, 2021.

Приложение А. Листинг программы

**Configuration.java**

package future.code.dark.dungeon.config;

public interface Configuration {

String GAME\_NAME="Dark Dungeon";

String MAP\_FILE\_PATH = "src/main/resources/maps/map.ber";

String RESULT\_FILE\_PATH = "src/main/resources/results/result.txt";

Boolean ENEMIES\_ACTIVE = true;

int MILLISECONDS\_PER\_FRAME = 40;

char WALL\_CHARACTER = '1';

char EXIT\_CHARACTER = 'E';

char LAND\_CHARACTER = '0';

char PLAYER\_CHARACTER = 'P';

char ENEMY\_CHARACTER = 'G';

char COIN\_CHARACTER = 'C';

Integer SPRITE\_SIZE = 64;

String PLAYER\_SPRITE = "src/main/resources/assets/hero/tile000.png";

String GHOST\_SPRITE = "src/main/resources/assets/ghost/tile000.png";

String WALL\_SPRITE = "src/main/resources/assets/land/wall.png";

String LAND\_SPRITE = "src/main/resources/assets/land/ground.png";

String EXIT\_SPRITE = "src/main/resources/assets/land/out.png";

String COIN\_SPRITE = "src/main/resources/assets/land/collectible.png";

String VICTORY = "src/main/resources/assets/victory.jpg";

String DEFEAT = "src/main/resources/assets/game\_over\_screen.jpeg";

String START = "src/main/resources/assets/menu/start.jpg";

String QUIT = "src/main/resources/assets/menu/quit.png";

}

**MouseController.java**

package future.code.dark.dungeon.controller;

import future.code.dark.dungeon.service.GameMaster;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import static future.code.dark.dungeon.config.Configuration.SPRITE\_SIZE;

public class MouseContoller implements MouseListener {

private final GameMaster gameMaster;

public MouseContoller(GameMaster gameMaster) {

this.gameMaster = gameMaster;

}

@Override

public void mouseClicked(MouseEvent e) {

if(this.gameMaster.getMenu()) {

if (e.getX() >= this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 3 &&

e.getY() >= this.gameMaster.getMap().getHeight() \* SPRITE\_SIZE / 5 &&

e.getX() <= this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 3 + this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 3 &&

e.getY() <= this.gameMaster.getMap().getHeight() \* SPRITE\_SIZE / 7 + this.gameMaster.getMap().getHeight() \* SPRITE\_SIZE / 5) {

this.gameMaster.setMenu(false);

} else if (e.getX() >= this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 3 &&

e.getY() >= this.gameMaster.getMap().getHeight() \* SPRITE\_SIZE / 2 &&

e.getX() <= this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 3 + this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 3 &&

e.getY() <= this.gameMaster.getMap().getHeight() \* SPRITE\_SIZE / 7 + this.gameMaster.getMap().getWidth() \* SPRITE\_SIZE / 2) {

System.exit(1);

}

}

}

@Override

public void mousePressed(MouseEvent e) {

}

@Override

public void mouseReleased(MouseEvent e) {

}

@Override

public void mouseEntered(MouseEvent e) {

}

@Override

public void mouseExited(MouseEvent e) {

}

}

**MovementController.java**

package future.code.dark.dungeon.controller;

import future.code.dark.dungeon.domen.Player;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import static future.code.dark.dungeon.domen.DynamicObject.Direction.DOWN;

import static future.code.dark.dungeon.domen.DynamicObject.Direction.LEFT;

import static future.code.dark.dungeon.domen.DynamicObject.Direction.RIGHT;

import static future.code.dark.dungeon.domen.DynamicObject.Direction.UP;

public class MovementController extends KeyAdapter {

private final Player player;

public MovementController(Player player) {

this.player = player;

}

@Override

public void keyPressed(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_UP -> player.move(UP);

case KeyEvent.VK\_DOWN -> player.move(DOWN);

case KeyEvent.VK\_LEFT -> player.move(LEFT);

case KeyEvent.VK\_RIGHT -> player.move(RIGHT);

}

}

@Override

public void keyReleased(KeyEvent e) {

}

}

**Coin.java**

package future.code.dark.dungeon.domen;

import future.code.dark.dungeon.config.Configuration;

public class Coin extends GameObject{

public Coin(int xPosition, int yPosition) {

super(xPosition, yPosition, Configuration.COIN\_SPRITE);

}

}

**DynamicObject.java**

package future.code.dark.dungeon.domen;

import future.code.dark.dungeon.config.Configuration;

import future.code.dark.dungeon.service.GameMaster;

public abstract class DynamicObject extends GameObject {

public DynamicObject(int xPosition, int yPosition, String imagePath) {

super(xPosition, yPosition, imagePath);

}

public enum Direction {

UP, DOWN, LEFT, RIGHT

}

protected void move(Direction direction, int distance) {

int tmpXPosition = getXPosition();

int tmpYPosition = getYPosition();

switch (direction) {

case UP -> tmpYPosition -= distance;

case DOWN -> tmpYPosition += distance;

case LEFT -> tmpXPosition -= distance;

case RIGHT -> tmpXPosition += distance;

}

if (isAllowedSurface(tmpXPosition, tmpYPosition)) {

xPosition = tmpXPosition;

yPosition = tmpYPosition;

}

}

public Boolean enemyMoveAvailuable(int x,int y){

return isAllowedSurface(x,y);

}

public Boolean collision(int x,int y){

return this.xPosition == x && this.yPosition ==y;

}

//если позиция не равна стене и выоду

private Boolean isAllowedSurface(int x, int y) {

if(GameMaster.getInstance().getMap().getMap()[y][x] == Configuration.EXIT\_CHARACTER && GameMaster.getInstance().getCoins().size()!=0)

return false;

return GameMaster.getInstance().getMap().getMap()[y][x] != Configuration.WALL\_CHARACTER;

}

}

**Enemy.java**

package future.code.dark.dungeon.domen;

import future.code.dark.dungeon.config.Configuration;

import future.code.dark.dungeon.service.GameMaster;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Enemy extends DynamicObject {

public static final int stepSize = 1;

public Enemy(int xPosition, int yPosition) {

super(xPosition, yPosition, Configuration.GHOST\_SPRITE);

}

public void move(Direction direction){

super.move(direction, stepSize);

Player player = GameMaster.getInstance().getPlayer();

if(collision(player.getXPosition(), player.getYPosition())){

player.dead = true;

}

}

public Boolean moveAvailuable(int x, int y){

return super.enemyMoveAvailuable(x,y);

}

}

**Exit.java**

package future.code.dark.dungeon.domen;

import static future.code.dark.dungeon.config.Configuration.EXIT\_SPRITE;

public class Exit extends GameObject{

public Exit(int xPosition, int yPosition) {

super(xPosition, yPosition, EXIT\_SPRITE);

}

public int getExitXPosition(){

return xPosition;

}

public int getExitYPosition(){

return yPosition;

}

}

**GameObject.java**

package future.code.dark.dungeon.domen;

import javax.swing.ImageIcon;

import java.awt.Graphics;

import java.awt.Image;

import static future.code.dark.dungeon.config.Configuration.SPRITE\_SIZE;

public abstract class GameObject {

private final Image image;

protected int xPosition;

protected int yPosition;

public GameObject(int xPosition, int yPosition, String imagePath) {

this.xPosition = xPosition;

this.yPosition = yPosition;

this.image = new ImageIcon(imagePath).getImage();

}

public int getXPosition() {

return xPosition;

}

public int getYPosition() {

return yPosition;

}

public void setXPosition(int xPosition) {this.xPosition = xPosition;}

public void setYPosition(int yPosition) {this.yPosition = yPosition;}

public void render(Graphics graphics) {

graphics.drawImage(image, xPosition \* SPRITE\_SIZE, yPosition \* SPRITE\_SIZE, null);

}

}

**Map.java**

package future.code.dark.dungeon.domen;

import future.code.dark.dungeon.util.FileUtils;

import javax.swing.ImageIcon;

import java.awt.Graphics;

import java.awt.Image;

import java.io.FileNotFoundException;

import java.util.List;

import java.util.stream.Collector;

import static future.code.dark.dungeon.config.Configuration.EXIT\_CHARACTER;

import static future.code.dark.dungeon.config.Configuration.EXIT\_SPRITE;

import static future.code.dark.dungeon.config.Configuration.LAND\_SPRITE;

import static future.code.dark.dungeon.config.Configuration.PLAYER\_CHARACTER;

import static future.code.dark.dungeon.config.Configuration.SPRITE\_SIZE;

import static future.code.dark.dungeon.config.Configuration.WALL\_CHARACTER;

import static future.code.dark.dungeon.config.Configuration.WALL\_SPRITE;

public class Map {

private static final Image wallImage = new ImageIcon(WALL\_SPRITE).getImage();

private static final Image landImage = new ImageIcon(LAND\_SPRITE).getImage();

private static final Image exitImage = new ImageIcon(EXIT\_SPRITE).getImage();

private final char[][] map;

public Map(String mapPath) throws FileNotFoundException {

List<String> lines = FileUtils.readFile(mapPath);

if (!validate(lines)) {

throw new RuntimeException("Map is not valid");

}

this.map = createMap(lines);

}

private char[][] createMap(List<String> lines) {

char[][] map = new char[lines.size()][lines.get(0).length()];

int rowNumber = 0;

for (String line : lines) {

map[rowNumber++] = line.toCharArray();

}

return map;

}

private Boolean validate(List<String> lines) {

char[] leftColumn = lines.stream().map(item -> item.toCharArray()[0]).collect(Collector.of(

StringBuilder::new,

StringBuilder::append,

StringBuilder::append,

StringBuilder::toString)).toCharArray();

char[] rightColumn = lines.stream().map(item -> item.toCharArray()[item.length()-1]).collect(Collector.of(

StringBuilder::new,

StringBuilder::append,

StringBuilder::append,

StringBuilder::toString)).toCharArray();

if (!containsOnlyWalls(lines.get(0).toCharArray()) ||

!containsOnlyWalls(lines.get(lines.size() - 1).toCharArray()) ||

!containsOnlyWalls(leftColumn) ||

!containsOnlyWalls(rightColumn)

) {

return false;

}

return containsExit(lines) && containsPlayer(lines) && lines

.stream().allMatch(line -> line.length() == lines.get(0).length());

}

private Boolean containsOnlyWalls(char[] line) {

for (char ch : line) {

if (ch != WALL\_CHARACTER) {

return false;

}

}

return true;

}

private Boolean containsExit(List<String> lines) {

for (String line : lines) {

for (char ch : line.toCharArray()) {

if (ch == EXIT\_CHARACTER) {

return true;

}

}

}

return false;

}

private Boolean containsPlayer(List<String> lines) {

for (String line : lines) {

for (char ch : line.toCharArray()) {

if (ch == PLAYER\_CHARACTER) {

return true;

}

}

}

return false;

}

public void render(Graphics graphics) {

for (int i = 0; i < map.length; i++) {

for (int j = 0; j < map[i].length; j++) {

if (map[i][j] == WALL\_CHARACTER) {

graphics.drawImage(wallImage, j \* SPRITE\_SIZE, i \* SPRITE\_SIZE, null);

} else {

graphics.drawImage(landImage, j \* SPRITE\_SIZE, i \* SPRITE\_SIZE, null);

}

}

}

}

public int getWidth() {

return map[0].length;

}

public int getHeight() {

return map.length;

}

public char[][] getMap() {

return map;

}

}

**Player.java**

package future.code.dark.dungeon.domen;

import future.code.dark.dungeon.config.Configuration;

import future.code.dark.dungeon.service.GameMaster;

public class Player extends DynamicObject {

private static final int stepSize = 1;

public boolean exited=false;

public boolean dead=false;

public Player(int xPosition, int yPosition) {

super(xPosition, yPosition, Configuration.PLAYER\_SPRITE);

}

public void move(Direction direction) {

super.move(direction, stepSize);

if(GameMaster.getInstance().getCoins().stream()

.anyMatch((coin -> collision(coin.xPosition,coin.yPosition)))){

GameMaster.getInstance().deleteCoin(this.xPosition,this.yPosition);

}

if(GameMaster.getInstance().getExit().stream()

.anyMatch((exit -> collision(exit.xPosition,exit.yPosition)))){

exited = true;

}

if(GameMaster.getInstance().getEnemies().stream()

.anyMatch(enemy -> collision(enemy.getXPosition(), enemy.getYPosition()))){

dead = true;

}

}

@Override

public String toString() {

return "Player{[" + xPosition + ":" + yPosition + "]}";

}

}

**GameMaster.java**

package future.code.dark.dungeon.service;

import future.code.dark.dungeon.config.Configuration;

import future.code.dark.dungeon.domen.Coin;

import future.code.dark.dungeon.domen.DynamicObject;

import future.code.dark.dungeon.domen.Enemy;

import future.code.dark.dungeon.domen.Exit;

import future.code.dark.dungeon.domen.GameObject;

import future.code.dark.dungeon.domen.Map;

import future.code.dark.dungeon.domen.Player;

import future.code.dark.dungeon.util.FileUtils;

import future.code.dark.dungeon.util.Vertex;

import javax.swing.\*;

import java.awt.\*;

import java.io.FileNotFoundException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.function.Consumer;

import java.util.stream.Collectors;

import static future.code.dark.dungeon.config.Configuration.\*;

import static future.code.dark.dungeon.domen.DynamicObject.Direction.\*;

public class GameMaster {

private static GameMaster instance;

private final Map map;

public Integer enemyCounter = 0;

public Integer endGameCounter = 0;

private List<GameObject> gameObjects;

Integer score=0;

Image victory = new ImageIcon(VICTORY).getImage();

Image defeat = new ImageIcon(DEFEAT).getImage();

Image start = new ImageIcon(START).getImage();

Image quit = new ImageIcon(QUIT).getImage();

private Vertex graph;

private boolean menu=true;

long startTime;

long endTime;

long duration;

public static synchronized GameMaster getInstance() {

if (instance == null) {

instance = new GameMaster();

}

return instance;

}

private GameMaster() {

try {

this.map = new Map(Configuration.MAP\_FILE\_PATH);

this.gameObjects = initGameObjects(map.getMap());

this.graph = Vertex.createGraph(map);

} catch (FileNotFoundException e) {

throw new RuntimeException(e);

}

}

private List<GameObject> initGameObjects(char[][] map) {

List<GameObject> gameObjects = new ArrayList<>();

Consumer<GameObject> addGameObject = gameObjects::add;

Consumer<Enemy> addEnemy = enemy -> {if (ENEMIES\_ACTIVE) gameObjects.add(enemy);};

for (int i = 0; i < map.length; i++) {

for (int j = 0; j < map[i].length; j++) {

switch (map[i][j]) {

case EXIT\_CHARACTER -> addGameObject.accept(new Exit(j, i));

case COIN\_CHARACTER -> addGameObject.accept(new Coin(j, i));

case ENEMY\_CHARACTER -> addEnemy.accept(new Enemy(j, i));

case PLAYER\_CHARACTER -> addGameObject.accept(new Player(j, i));

}

}

}

return gameObjects;

}

public void renderFrame(Graphics graphics) {

if(menu){

graphics.fillRect(0,0, this.getMap().getWidth() \* SPRITE\_SIZE, this.getMap().getHeight() \* SPRITE\_SIZE);

graphics.drawImage(start,

this.getMap().getWidth() \* SPRITE\_SIZE/3,

this.getMap().getHeight() \* SPRITE\_SIZE/5,

this.getMap().getWidth()\*SPRITE\_SIZE/3,

this.getMap().getHeight()\*SPRITE\_SIZE/7,

null);

graphics.drawImage(quit,

this.getMap().getWidth() \* SPRITE\_SIZE/3,

this.getMap().getHeight() \* SPRITE\_SIZE/2,

this.getMap().getWidth()\*SPRITE\_SIZE/3,

this.getMap().getHeight()\*SPRITE\_SIZE/7,

null);

startTime = System.currentTimeMillis();

}

else if(getPlayer().dead){

endTime=System.currentTimeMillis();

if(duration == 0) duration = endTime - startTime;

graphics.fillRect(0,0, this.getMap().getWidth() \* SPRITE\_SIZE, this.getMap().getHeight() \* SPRITE\_SIZE);

graphics.drawImage(defeat,this.getMap().getWidth() \* SPRITE\_SIZE/6,0,null);

endGameCounter++;

if(endGameCounter == 120){

try {

FileUtils.writeFile(RESULT\_FILE\_PATH,(int)duration/1000,false);

} catch (FileNotFoundException e) {

throw new RuntimeException(e);

}

duration=0;

menu=true;

getPlayer().dead=false;

Player pl = getPlayer();

for (int i = 0; i < map.getHeight(); i++) {

for (int j = 0; j < map.getWidth(); j++) {

switch (map.getMap()[i][j]) {

case PLAYER\_CHARACTER -> {

pl.setXPosition(j);

pl.setYPosition(i);

}

}

}

}

this.gameObjects = initGameObjects(map.getMap());

this.graph = Vertex.createGraph(map);

setPlayer(pl);

endGameCounter = 0;

}

}

else if(!getPlayer().exited) {

getMap().render(graphics);

getStaticObjects().forEach(gameObject -> gameObject.render(graphics));

getEnemies().forEach(gameObject -> gameObject.render(graphics));

getPlayer().render(graphics);

graphics.setColor(Color.WHITE);

graphics.drawString(getPlayer().toString(), 10, 20);

graphics.drawString("Collected coins: " + score.toString(), 120, 20);

graphics.drawString("Coins lost: " + getCoins().size(), 240, 20);

enemyCounter++;

if(enemyCounter==30){

for(Enemy enem: getEnemies()) {

Vertex way = graph.findNextWay(this.getPlayer(),enem, getInstance().getEnemies());

if(way==null && enem.collision(getPlayer().getXPosition(), getPlayer().getYPosition())){

getPlayer().dead = true;

break;

}

if(way!=null) {

if (enem.getXPosition() < way.getX()) enem.move(RIGHT);

else if (enem.getXPosition() > way.getX()) enem.move(LEFT);

else if (enem.getYPosition() < way.getY()) enem.move(DOWN);

else enem.move(UP);

}

}

enemyCounter=0;

}

}

else{

endTime=System.currentTimeMillis();

if(duration == 0) duration = endTime - startTime;

graphics.fillRect(0,0, this.getMap().getWidth() \* SPRITE\_SIZE, this.getMap().getHeight() \* SPRITE\_SIZE);

graphics.drawImage(victory,this.getMap().getWidth() \* SPRITE\_SIZE/6,0,null);

endGameCounter++;

if(endGameCounter == 120){

try {

FileUtils.writeFile(RESULT\_FILE\_PATH,duration/1000,true);

} catch (FileNotFoundException e) {

throw new RuntimeException(e);

}

duration=0;

menu=true;

getPlayer().exited=false;

Player pl = getPlayer();

for (int i = 0; i < map.getHeight(); i++) {

for (int j = 0; j < map.getWidth(); j++) {

switch (map.getMap()[i][j]) {

case PLAYER\_CHARACTER -> {

pl.setXPosition(j);

pl.setYPosition(i);

}

}

}

}

this.gameObjects = initGameObjects(map.getMap());

this.graph = Vertex.createGraph(map);

setPlayer(pl);

endGameCounter = 0;

}

}

}

public Player getPlayer() {

return (Player) gameObjects.stream()

.filter(gameObject -> gameObject instanceof Player)

.findFirst()

.orElseThrow();

}

private void setPlayer(Player player){

gameObjects.removeIf((gameObject -> gameObject instanceof Player));

gameObjects.add(player);

}

private List<GameObject> getStaticObjects() {

return gameObjects.stream()

.filter(gameObject -> !(gameObject instanceof DynamicObject))

.collect(Collectors.toList());

}

public List<Enemy> getEnemies() {

return gameObjects.stream()

.filter(gameObject -> gameObject instanceof Enemy)

.map(gameObject -> (Enemy) gameObject)

.collect(Collectors.toList());

}

public List<Coin> getCoins(){

return gameObjects.stream()

.filter(gameObject ->gameObject instanceof Coin)

.map(gameObject -> (Coin) gameObject)

.collect(Collectors.toList());

}

public List<Exit> getExit(){

return gameObjects.stream()

.filter(gameObject -> gameObject instanceof Exit)

.map(gameObject -> (Exit) gameObject)

.collect(Collectors.toList());

}

public void deleteCoin(int x, int y){

this.gameObjects.removeIf(coin-> coin instanceof Coin && coin.getXPosition() == x && coin.getYPosition() == y);

score++;

}

public Map getMap() {

return map;

}

public void setMenu(boolean menu) {

this.menu = menu;

}

public boolean getMenu(){return menu;}

}

**FileUtils.java**

package future.code.dark.dungeon.util;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.Scanner;

public class FileUtils {

public static List<String> readFile(String path) throws FileNotFoundException {

Scanner scanner = new Scanner(new File(path));

List<String> lines = new ArrayList<>();

while (scanner.hasNextLine()) {

lines.add(scanner.nextLine());

}

return lines;

}

public static void writeFile(String path, long time, boolean win) throws FileNotFoundException {

String result = "Defeat";

long minutes = time / 60;

Date date = new Date();

if(win) result = "Win";

try{

FileWriter fileWriter = new FileWriter(path,true);

fileWriter.write(date+"|| Result: " + result + " Time: " + minutes + "m " + time % 60+"s\n");

fileWriter.close();

} catch (IOException e) {

System.out.println("Ошибка при записи в файл");

throw new RuntimeException(e);

}

}

}

**Vertex.java**

package future.code.dark.dungeon.util;

import future.code.dark.dungeon.domen.Enemy;

import future.code.dark.dungeon.domen.Map;

import future.code.dark.dungeon.domen.Player;

import java.util.\*;

import static future.code.dark.dungeon.config.Configuration.\*;

public class Vertex {

private int x;

private int y;

private Vertex next;

private Vertex[] adjVerts;

private int distId;

private Vertex[] shortPath;

public Vertex(int x,int y, Vertex[] adjVerts, int distId) {

this.x = x;

this.y = y;

this.adjVerts = adjVerts;

this.distId = distId;

}

public static Vertex createGraph(Map map){

char[][] array = map.getMap();

int k=0;

for(int i = 1; i < array.length; i++){

for(int j = 1; j < array[i].length; j++){

if(array[i][j] != WALL\_CHARACTER) k++;

}

}

Vertex[] verts = new Vertex[k];

k=0;

for(int i=1;i<array.length;i++){

for(int j=1;j<array[i].length;j++){

if(array[i][j] != WALL\_CHARACTER){

verts[k] = new Vertex(j,i,null,k);

k++;

}

}

}

for(int i=0;i<verts.length;i++){

if(i!=verts.length-1)

verts[i].next = verts[i+1];

}

for(int i=0;i<verts.length;i++) {

Vertex vert = verts[i];

Vertex[] adj = new Vertex[4];

k=0;

int x=vert.x;

int y=vert.y;

if(array[y][x-1] != WALL\_CHARACTER) {

Vertex temp = Arrays.stream(verts).filter(n->n.y==y && n.x==x-1).findFirst().get();

adj[k] = temp;

k++;

}

if(array[y-1][x] != WALL\_CHARACTER) {

Vertex temp = Arrays.stream(verts).filter(n->n.y==y-1 && n.x==x).findFirst().get();

adj[k] = temp;

k++;

}

if(array[y][x+1] != WALL\_CHARACTER) {

Vertex temp = Arrays.stream(verts).filter(n->n.y==y && n.x==x+1).findFirst().get();

adj[k] = temp;

k++;

}

if(array[y+1][x] != WALL\_CHARACTER) {

Vertex temp = Arrays.stream(verts).filter(n->n.y==y+1 && n.x==x).findFirst().get();

adj[k] = temp;

k++;

}

vert.adjVerts = Arrays.copyOf(adj, k);

}

return verts[0];

}

public Vertex findNextWay(Player player, Enemy enemy, List<Enemy> otherEnemies){

Vertex tmp = this;

int k=0;

while(tmp!=null){

tmp.shortPath=null;

k++;

tmp = tmp.next;

}

tmp = this;

int[] dists=new int[k];

Arrays.fill(dists,-1);

tmp = this;

while(tmp!=null){

if(tmp.x==enemy.getXPosition() && tmp.y==enemy.getYPosition())

break;

tmp=tmp.next;

}

if(player.dead) return tmp;

Vertex[] visVerts=null;

Queue<Vertex> libQueue = new LinkedList<Vertex>();

libQueue.add(tmp);

visVerts=new Vertex[1];

visVerts[0]=tmp;

while(!libQueue.isEmpty()){

Vertex newVert = libQueue.poll();

if(newVert.adjVerts!=null) {

for (int i = 0; i < newVert.adjVerts.length; i++) {

final Vertex vertex = newVert.adjVerts[i];

if (Arrays.stream(visVerts).noneMatch(n -> n == vertex) && otherEnemies.stream().noneMatch(n->n.collision(vertex.x, vertex.y))) {

libQueue.add(vertex);

Vertex[] tempVerts = new Vertex[visVerts.length + 1];

tempVerts = Arrays.copyOf(visVerts, visVerts.length + 1);

tempVerts[visVerts.length] = vertex;

visVerts = tempVerts;

dists[vertex.distId]=dists[newVert.distId]+1;

Vertex[] shortPath;

if(newVert.shortPath!=null)

shortPath = Arrays.copyOf(newVert.shortPath, newVert.shortPath.length+1);

else shortPath = new Vertex[1];

shortPath[shortPath.length-1] = vertex;

vertex.shortPath = shortPath;

}

}

}

}

tmp = this;

while(tmp!=null){

if(tmp.x==player.getXPosition() && tmp.y==player.getYPosition())

break;

tmp=tmp.next;

}

if(tmp.shortPath==null) return null;

return tmp.shortPath[0];

}

public int getX(){return this.x;}

public int getY(){return this.y;}

}

**GameFrame.java**

package future.code.dark.dungeon;

import future.code.dark.dungeon.controller.MouseContoller;

import future.code.dark.dungeon.controller.MovementController;

import future.code.dark.dungeon.service.GameMaster;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.Timer;

import java.awt.Graphics;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import static future.code.dark.dungeon.config.Configuration.MILLISECONDS\_PER\_FRAME;

import static future.code.dark.dungeon.config.Configuration.SPRITE\_SIZE;

public class GameFrame extends JPanel implements ActionListener {

private final GameMaster gameMaster;

public GameFrame(JFrame frame) {

Timer timer = new Timer(MILLISECONDS\_PER\_FRAME, this);

this.gameMaster = GameMaster.getInstance();

frame.setSize(gameMaster.getMap().getWidth() \* SPRITE\_SIZE, gameMaster.getMap().getHeight() \* SPRITE\_SIZE);

frame.setLocationRelativeTo(null);

timer.start();

frame.addKeyListener(new MovementController(gameMaster.getPlayer()));

frame.addMouseListener(new MouseContoller(gameMaster));

}

@Override

public void paint(Graphics graphics) {

gameMaster.renderFrame(graphics);

}

@Override

public void actionPerformed(ActionEvent e) { // Always triggered by Timer

repaint();

}

}

**Main.java**

package future.code.dark.dungeon;

import future.code.dark.dungeon.config.Configuration;

import javax.swing.JFrame;

public class Main {

public static void main(String[] args) {

JFrame frame = new JFrame(Configuration.GAME\_NAME);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(false);

frame.add(new GameFrame(frame));

frame.setVisible(true);

}

}